6

Time-Sharing Computer Systems

Speaker	JOHN MCCARTHY Associate Professor of Communications Sciences Massachusetts Institute of Technology
Discussants	JOHN W. MAUCHLY President Mauchly Associates Inc. GENE M. AMDAHL Manager, Advanced Systems Design IBM Data Systems Division
Moderator	EMANUEL R. PIORE Vice President for Research and Engineering International Business Machines Corporation

MCCARTHY. I am going to discuss the important trend in computer design toward time-sharing computer systems. By a time-sharing computer system I shall mean one that interacts with many simultaneous users through a number of remote consoles. Such a system will look to each user like a large private computer. The new applications that time sharing will make possible will be of as much additional benefit to science and management as resulted from the introduction of the stored-program digital computer.

First I shall discuss the uses of a large private computer, and then I shall discuss how the same effect can be achieved through time sharing. I shall also touch upon the requirements that time-sharing systems impose upon computer design. Some part of what I am going to say arose in connection with the work

of the Long Range Computer Study Group here at M.I.T. The material I shall present on computer-system design was developed jointly with Marvin Minsky. I also want to acknowledge the stimulating effect of discussions with Professor Herbert M. Teager and Dr. F. J. Corbató, who are developing time-sharing systems for the IBM 7090 at the M.I.T. Computation Center.

Some of the ideas concerning time-sharing systems go back quite a way. The first paper on the subject that I know of was written by Christopher Strachey and was delivered at the Paris International Conference on Information Processing.[1] A more recent paper is by Licklider.[2] The subject was also touched on in the lectures by Kemeny and Perlis. In 1945 Vannevar Bush discussed a system for personal information retrieval called Memex, which probably requires a computer system of the kind that I am going to discuss for its realization.[3]

A Private Computer

Why should anyone want a private computer? The reason has to do with the interaction between user and computer. In theoretical terms, a communication channel can be characterized not only by its one-way capacity, but also by its two-way capacity, a concept that is a little harder to define formally. Two-way capacity is the number of messages per second which the channel can interchange, given that the content of the leaving message depends on the content of the entering message. The advantage of a private computer is that it makes possible a large two-way communication capacity.

What the user wants is a computer that he can have continuously at his beck and call for long periods of time. Of course, computer applications differ in the amount of interaction required between user and machine (interaction can be measured by how often messages have to be exchanged between user and machine in order to complete the task). The earliest applications of computers were those in which there was little two-way interaction. Programs could be run for long periods of time without any user involvement. For instance, in the com-

putation of mathematical tables (one of the earliest applications), you knew the tables that you wanted to compute, so you designed the machine and wrote the program and then let the thing go. Eventually the computer and the program and, unfortunately, also the tables became simultaneously obsolete.

There are still many applications that require long periods of straight computing, even with present computers. Examples are numerical weather prediction, which is still a very substantial task and promises to become even more so as models become more sophisticated, and calculations of the distribution of energy in nuclear reactors. In my remarks, however, I shall concentrate on applications, present and potential, that require frequent interaction between user and machine.

Debugging

One such application is programming. This application forces itself upon the user and requires more frequent interaction between man and machine than is presently available. Even people who are trying to program for conventional applications find themselves suffering from the lack of two-way interaction. If a program is at all complicated (and weather prediction programs are complicated), it will be built up from subprograms that require extensive testing singly and in combination. Scores of runs may be necessary to find and correct all the errors in an extensive program. Nevertheless, the amount of computer time consumed in a single run may be quite small. In fact, neglecting input-output time, only a few milliseconds may be required to detect an error in a straight-line program. Some errors, like misprints, can be corrected very quickly once the machine detects them; a programmer will get very annoyed if he has to wait a day in order to fix a misplaced comma. Other errors are more complicated, and the user may have to study his results for a day or two before he finds the trouble.

Two Present Systems of Computer Operation

There are two common ways of operating a computer for debugging programs. Both are used at M.I.T.: one by users of

the TX–0 computer, and the other by the users of the IBM 709 computer at the Computation Center.

The TX–0 is a small computer that is operated very informally. Users sign up a week in advance for blocks of time of from one to several hours. During his block of time the user has a complete monopoly of the computer. He tests his program, makes necessary corrections, retests his program, and continues to make corrections. While the user is thinking about his results, the computer stands idle. If the source of trouble is hard to find, the user can waste his remaining allotment of time for the week. On the other hand, if the trouble is trivial, he can fix it immediately and continue. The TX–0 has about forty users, of whom approximately twenty-five are active in a typical week. If the TX–0 were a much larger computer, and if it were operated in the same manner as at present, the number of users who could be accommodated would still be about the same.

The IBM 709 at the M.I.T. Computation Center is a larger computer and is operated according to a different principle. Users prepare their programs and data on punched cards and deposit them in a file in a run preparation room. Machine operators transfer the programs from the punched cards to magnetic tape in batches. The programs are processed one after another by the computer, and the results of the computation are written on tape; these tapes are printed on a tape-to-printer machine, and the results are placed in the user's file. The advantage of this system is that there is no delay between individual users, and the machine is not idle while the user is thinking. By the nature of the system, a delay of two or three hours is implied between the time the cards for a job are submitted and the time that results come back. Since the Computation Center is overcrowded, however, the delay usually is more like one or two days; it is at its worst at thesis time, the end of the school year. Because the machine never waits while the user thinks, the computer can do from 125 to 160 jobs a day. The overcrowding results from the approximately 500 programmers who desire active use of the computer.

Neither the TX–0 system nor the 709 system is completely satisfactory. The TX–0 system is unsatisfactory because the user has to sign up well in advance and cannot go on whenever he is ready; in addition, he does not have time to think. The 709 system is unsatisfactory because of the delays in getting results back. Programming is feasible in both systems. However, the intellectual depth of projects that can be undertaken is limited by the time required to debug even relatively simple procedures. This has a very bad effect on student theses. When an ambitious student proposes to undertake something substantial, his adviser often must say, "I don't believe you can finish in time for a thesis." I have several students this semester who are in this kind of trouble. From the user's point of view the solution clearly is to have a private computer.

Man-Machine Interaction

What I have said so far has been about problems in programming that are more or less independent of what the programs are about. Now I should like to discuss applications that have an essential dependence on rapid interaction between man and machine. This topic is more speculative because these applications are undeveloped for lack of suitable machines. The first application that I should like to discuss is computing with symbolic expressions, a field in which I have worked. It is entirely feasible to have a computer manipulate formulas in the manner of a mathematician or a scientist doing theoretical work. For example, there are programs for symbolic differentiation, integration, solution of equations, simplification of algebraic expressions, the solution of differential equations, and a number of other processes. However, one rarely makes a two-hour run with symbolic calculations. If one has an integral to evaluate, even if an integration program is available, it is much easier to ask somebody (if you are not good at this sort of thing yourself) than it is to get out the program and apply it. This leads me to believe that some of the applications of symbolic calculation depend on having what is essentially a private computer, so that when you decide that you have a symbolic cal-

culation to make, you can simply type in what you want done.

Another very important application is the use of the computer as a teacher. There has been a considerable amount of work done on teaching machines. These machines are generally of a rather simple character. A fixed sequence of questions is presented to the student, who tries to answer. When he gets the right answer, he goes on to the next question. The advantages over conventional methods are that the student can work at his own pace, and immediate reinforcement is provided for a correct answer (according to many psychologists this is very important). Present teaching machines, however, give the questions in a fixed order that does not depend on the student's answers to earlier questions. These machines cannot react to an answer other than by fitting it into one of a number of predefined categories, and furthermore, they are very limited in the amount of information (in convenient form) which they can provide the teacher on the student's progress.

Using a computer as a teacher can overcome these limitations. The decision of which information to give or which question to ask can depend on previous performance in whatever way the instructor wants. New questions can be generated to cover points where the student requires additional work. With a large class of students, each one progressing at his own pace, the instructor can be given reports identifying students who require additional help or who are doing so well as to deserve special attention and recognition. It will not be easy, however, to program a computer to take full advantage of all this flexibility.

A third application is to provide the services envisaged in Professor Kemeny's lecture in this series on "A Library for 2000 A.D." It seems that except for the incorporation of a large mass of documents, Professor Kemeny's library, and then some, can be realized, not by the year 2000, but by the year 1965.

The last specific application that I shall mention concerns interaction with laboratory apparatus in real time. The parts of laboratory apparatus that serve merely to transform in-

formation are at present very expensive to construct and tedious to debug. For a great many purposes they can be replaced conveniently by two-way connections to a computer.

I have mentioned some specific applications of a computer which require considerably more two-way interaction capability than we have at present. I could add to this by saying some more general things about the possible role of a computer as an intellectual servant. A computer can carry out any intellectual process that we can program or describe precisely. Our ability to describe processes precisely, however, is still very limited, as those of us who work on artificial-intelligence problems will certainly agree. Nevertheless, some processes of scientific research already can be reduced to routine form, and if we had a computer with which efficient two-way interaction were possible, we should be able to extend greatly our scientific capability. As our ability to understand intellectual processes expands, we shall be able to apply the computer to more and more tasks.

Advantages of a Large Computer

I should now like to discuss the question of a large computer, where by large computer I mean one with very large primary storage and very large secondary storage. The amount of secondary storage available has a simple meaning. It determines what fraction of the computer culture, so to speak, is directly accessible to the users of the computer. The implications of having a large primary storage are more subtle.

The limitations on what it has been possible to do with computers so far depend partly on the computers themselves, but even more on our ability to program. Each computer has an order code that is closely connected with the way the computer was designed, and the computer is able to obey programs that are written in this order code (also known at the *machine's language*). As you undoubtedly know, this does not mean that the user must necessarily describe his procedure in machine language. One of the intellectual procedures that the computer can be programmed to perform is the translation of procedures

into machine language from source languages in which it is more convenient to write the procedures. A number of convenient source languages have been devised, and compilers and interpreters have been written which enable a computer to obey a program written in these languages. Among these are FORTRAN, which was developed by IBM for describing numerical procedures; ALGOL, which is a proposed international standard language for describing numerical procedures; COBOL, which is a proposed United States standard language for describing file maintenance and other business data processing proeedures; APT, a language that was devised at M.I.T. for programming the numerical control of machine tools; and LISP, a language that our group devised for describing computations with symbolic expressions.

Before computers there really was little need for convenient ways of formally describing procedures, and not much progress was made in developing such ways. At present, programming languages are developing rapidly, and there even has been work on constructing a mathematical theory of computation. Nevertheless, programming difficulties still cause most of the delay in preparing computers to carry out complicated procedures. If we are going to have the interaction ability that I described earlier, then we must have complex programming systems for translating procedures from source languages into machine language. If the user, sitting at his console, is going to be able to have symbolic computations performed for him, these procedures have to be at his beck and call. Ideally, the whole existing programming culture, including elaborate programming systems, should be present and readily accessible.

The main reason for needing a large primary memory is that the procedures which we can program are getting ever more elaborate. In fact, the frontier in computation is complication. An indication of this is the program for symbolic integration developed by Jim Slagle here at M.I.T. which barely fitted in the 32,768-word memory of the IBM 709. Much of the time required for developing this program was consumed in making the fit.[4] Now, symbolic integration is one of the simpler mathematical processes and occurs as a minor component in more interesting symbolic computations. We shall soon be writing programs that require an order of magnitude more memory space.

Time Sharing

I should like to go on now to consider how the private computer can be achieved. It is done by time sharing a large computer. Each user has a console that is connected to the computer by a wired channel such as a telephone line. The consoles are of two kinds, one cheap and the other better but more expensive. The cheap console is simply an electric typewriter that is used for both input and output.

Consider the operation of such a system as it appears to the user of the typewriter console. When the user wants service, he simply starts typing in a message requesting the service. The computer is always ready to pay attention to any key that he may strike, just as the telephone system is always ready for you to lift the receiver off the hook. As soon as the key is depressed on the typewriter, a signal is sent to the computer. The effect of the signal is to make the computer interrupt the program after the current instruction has been executed and jump temporarily to a program that determines what the typewriter wants. Most characters are at the beginning or middle of a message, so all the computer usually has to do is store the character or bring out a new character in case of output. If more than one typewriter requests attention simultaneously, a queue is formed. A very large number of typewriters can be handled without the computer taking more than a small amount of time away from whatever other tasks it is doing. In fact, a computer of the speed of the IBM 7090 could handle three thousand typewriters simultaneously.

Eventually a message will be completed, and the computer will have to take some kind of action other than merely storing characters. The following are examples of the kinds of action that may be required. First, the message may be a request for information from primary or secondary storage to be typed out.

Second, the message may be an expression to be integrated. Third, the message may be a statement in a program that, if formed according to the rules of the programming language, is to be added to a list of statements waiting to be translated when the program is complete. Fourth, the message may be a command to allocate an area in primary storage, or to transfer certain information from the user's files to primary storage for later action, or to transfer information from primary storage to the user's files. Fifth, the message may be a request to run a program taken from the typewriter, from the user's files, and from public subroutines. This fifth example takes the most computer time; the other examples involve specific, minor items that the computer can handle quickly.

At any moment some of the typewriters will be inactive, some will be in the middle of entering messages into the computer, some will be in the process of typing out characters, and some will be in a status of wanting programs run. There are several ways to handle the situation of a number of programs that simultaneously want to be run. I am going to discuss only the simplest way: the round-robin system. Each program is run for a definite period of time, which I shall call one quantum. If the program is not completed in this quantum, a time clock interruption occurs, and control passes to the next program in the round robin. When the round robin is completed, control returns to the first program, and so on.

How long should the quantum of time be? One answer is that the amount of time for the complete round robin should be just less than a human reaction time, say, one-tenth of a second. If this is so, then a user whose run requires less than one quantum of computer time will appear to get instantaneous service. The round-robin system is susceptible to various modifications. For example, very long programs may be postponed to slack hours in order to save memory space, and programs of very high priority can be given more than one quantum per round. A stop button would be an important part of the typewriter console, since it is very easy for a user to initiate an output that is much longer than he cares to have. There must be

some way of saying, "No, stop, and pay attention to me again."

There has been extensive experience with the use of computers with typewriters as their main input-output device. The main limitation of a typewriter is that it takes too long to produce ouput. For example, the report of an assembly or compilation may take as much as twenty minutes to produce. In our system this can be improved a little bit by having high-speed printing facilities available in the computation center, so that if a user will tolerate having his output later, he can have it printed at the center and sent to him.

The more expensive console that I mentioned can include a cathode-ray-tube unit on which the computer can display pictures and text. Such a unit requires some kind of temporary storage like a tape loop or drum to avoid the computer's spending excessive time maintaining the display. Another device that has been found very useful is a light pen. This simply consists of a photocell that the user can hold against the face of the cathode-ray tube to designate a point for the computer. Other analog or digital input-output connections are desirable if the computer is connected to laboratory apparatus.

This is one example of a time-sharing system. There are a large number of users. Each user has his own console, gets service from the computer whenever he desires it, and has the computer maintain his files for him.

Computer Requirements for Time Sharing

Time sharing is technically feasible on a small computer, but the full advantages of rapid man-machine interaction require elaborate programming languages and elaborate program control. Because programs may be called on by many users and may do only relatively short pieces of work between human interactions, it is uneconomical to have to shuttle them back and forth continually to and from secondary storage. Therefore, there is a requirement for a large primary memory. How large, we are not sure. The amount of permanent program that we should want to have continuously available might be some

100,000 words, given the present state of programming services, and this suggests something on the order of one million words of directly addressable core memory for a time-sharing system. The development of new programming services could increase this requirement considerably.

What are the other requirements for a time-sharing system? Present order codes of the computer appear adequate, and the main features of present single-address computers require no important variables. Some new features will be needed, however.

One requirement is for an interruption system to handle errors as well as input and output. An interruption system works in the following way. Upon a special condition, such as an erroneous instruction to divide by zero or an input-output unit requesting attention, the next instruction in sequence is not executed. Instead, the computer takes its next instruction from a specified location, determined by the condition that caused the interruption. The program to which control is transferred deals with the condition, and if the interruption is not due to a program error, returns control after it is done to the place where the interruption arose.

Another requirement is for completely nonstop operation. If the computer were to stop with ten or fifteen people typing, it would be very difficult to retrieve the situation. The operator would not know what the computer was doing or for whom. Almost all present computers either have certain instructions that can cause a stop, or have hang-up features when certain input-output signals are absent. A proper computer ought not to rely on anyone or anything for signals on schedule. It certainly should not rely on the users to program correctly, nor should it rely on input-output units to provide signals. In the case of a tape unit which fails to provide a signal saying that a character is ready, the computer could wait a prescribed time; if the signal were not forthcoming, it could interrupt to print out a complaint to the operators.

Another requirement has to do with erroneous programs that must be prevented from damaging other programs. This is best

handled by means of boundary registers and input-output interlocks. When the computer is executing a program, it must interrupt if the program attempts any memory references outside its allotted region, and it must also interrupt if the program attempts to use any instructions that activate input-output units. Input-output units should be activated only by system subroutines. Also, programs that get into endless loops must be prevented from wasting computer time. This can be accomplished by an alarm clock that interrupts a program after a quantum of time has been consumed and causes the user to be informed if his use becomes excessive.

An intricate problem arises from the fact that when a program ceases to be active it leaves an odd-sized hole in memory. It should be possible to move other programs down to fill the holes so as to provide contiguous blocks of space for new programs and data. This is facilitated by having a relocation register whose contents are added to addresses after they leave the central processing unit and before they get to memory. Another problem derives from the possibility that a system routine, such as a compiler, which is executing a program for one user, may be interrupted after a quantum of time to do another job for a different user. In order that this be feasible without confusion, it is necessary that system routines use temporary storage in blocks belonging to the user and not to themselves. Thus when program control returns to the first user, his temporary storage block is unaltered, and the system routine can take up where it left off.

The final requirement is for secondary storage large enough to maintain the users' files so that users need not have separate card or tape input-output units. For an institution like M.I.T., if only programs and data are stored (and not the M.I.T. library), then something like fifty million computer words appear adequate.

These computer requirements for time-sharing systems are not especially subtle, but the computers coming out nowadays seem to have a tendency to neglect many of them. For one thing, new computers usually are not able to address a very

large memory. Their order codes generally do not provide for memories beyond a certain size.

Multisequence Computers

I want to discuss certain economic considerations that apply to computers with very large memories. A million-word memory with reasonable speed of operation may cost six million dollars. Central processing units, on the other hand, are cheaper, so we have the situation of a very large and expensive memory coupled with a much cheaper central processing unit. Is it not possible to use this large memory more efficiently and to speed up the whole system by spending more money on the central processing unit? One example of a computer that attempts to obtain a higher speed than a straightforward design would allow, is the IBM STRETCH with 2-microsecond memory. The extra speed is achieved by means of a look-ahead mechanism that simultaneously picks up several instructions and, if possible, the data needed by these instructions. Unfortunately, it appears that the effective operation of such a look-ahead system requires that the machine designers anticipate how the machine is to be used. A failure to anticipate means that the computer will not be as fast as the designers hoped.

Some of us have worked on another scheme for obtaining higher speed in a computer system. The idea is to build a system of several separate central processing units and several separate memories, in general more memories than processing units. The central processing units all have the same order code and each has the ability to address the whole memory. Boundary registers ensure that a processing unit does not transgress the limits of memory set for it at a given time. When different processors address different memory boxes, simultaneous memory references occur, but when two processors request the same memory box, a device called the arbiter simply holds up one of them for one unit of time.

We propose that successive addresses be in separate memory boxes. Based on this concept, it is interesting to note what happens when two or more processors attempt to obey the

same program at the same time. This can happen when the program is a system routine, such as a compiler, that the processors are running for different users. Since the processors are working for different users, they will have different temporary storage areas, as I explained before, and there will not be any conflict there. However, there might be a conflict in getting instructions. Suppose, for example, that two processors ask for the same memory box. Then one of them is held up by the arbiter and hence falls a step behind the other one. As long as the processors execute instructions from successive addresses, they will not have any further conflicts. As soon as one of the processors executes a jump instruction, however, there is a certain probability that there will again be a conflict and that one processor will be delayed.

In a time-sharing system a processor executes a quantum of program for a user and then continues in the round robin to pick up the next user in line. Unless the number of users is a multiple of the number of processors, the next time a user's turn comes around, a different processor executes his program. Thus we see that there is no particular relationship between the number of users and the number of processors, nor is there any assignment of particular processors to particular users. The number of processors in a balanced system depends on the relative costs of processors and memory, and on the size of memory required.

Serial Processors

The relative cost of computer subsystems suggested an interesting system to Ed Fredkin. It seems that 5-microsecond cycle memory is now available at a reasonable price. Logical hardware, however, also at a reasonable price, is feasible at 10 megacycles per second. This means that the time required for one unit of logic is of the order of one-fiftieth of the memory cycle. Fredkin's idea is to use a serial processor in such a system. This is a processor in which the arithmetic registers are delay lines and the arithmetic is performed serially. Thus, in an addition the bits in the word stream serially through the

adder. A serial processor is much cheaper than the parallel processors currently used in large computer systems.

Computing as a Public Utility

In concluding I should like to say a word on management and the computer of the future. At present, computers are bought by individual companies or other institutions and are used only by the owning institution. If computers of the kind I have advocated become the computers of the future, then computation may someday be organized as a public utility, just as the telephone system is a public utility. We can envisage computing service companies whose subscribers are connected to them by telephone lines. Each subscriber needs to pay only for the capacity that he actually uses, but he has access to all programming languages characteristic of a very large system.

The system could develop commercially in fairly interesting ways. Certain subscribers might offer services to other subscribers. One example is weather prediction. A weather-predicting company that is a subscriber to a central computer predicts the weather but keeps the predictions in its private files. If you subscribe to its service, your programs can gain access to these files. You may even have weather-predicting programs run for your benefit to answer your own particular questions. Other possible services include those specifically connected with computing, such as programming services. Some subscribers perhaps might rent the use of their compilers. Other subscribers might furnish economic predictions. The computing utility could become the basis for a new and important industry.

Panel Discussion

MAUCHLY. Some people like a debate, but since in general I agree with what McCarthy has said, I cannot start one. Instead I wish to observe that if we are going to talk about future trends in computers, a look at the past could be helpful. For instance, the first electronic computer of large-scale, digital type

had some features that are just now appearing in other computers. This computer, the ENIAC, was able to read, write, and compute all at the same time. This was accomplished by a combination of equipment which we then described as the largest IBM plugboard in the world. The ENIAC used an IBM card reader and an IBM card punch. The programming was not stored as is customary nowadays, but was set up by cables, patch cords, switches, and similar things. As cumbersome as it sounds, the computer could operate rather fast. How did we do that? By making the computer operate in parallel fashion. The ENIAC was capable of something that is now coming back into vogue: multiprogramming. The main difference today is that we get much more capacity at a lesser cost. The departure from a parallel computer, which was able to do many separate problems simultaneously at rather high speeds, was made in an effort to achieve something more economical, similar to the motivation for the serial processor described at the end of McCarthy's talk. The more we learn about how to make hardware perform faster, the more we return to our efforts to perform larger and larger calculations at cheaper and cheaper rates. We appear to be traversing the full circle back to doing many things in parallel. If that represents a trend, it seems to be a circular one. Perhaps all we can deduce from this is that we tend to go around in circles.

There is another subject that I want to touch on here, and that is the subject of teaching machines. Perhaps this is not a new idea, but it seems to me that what we would really like to have is a learning-teaching machine. Almost any good teacher claims, and usually claims truthfully, that in the process of teaching he learns something. Yet practically all of the teaching machines that I have heard discussed so far are ones that are fixed-program. In other words, data from the student may feed back to the program of the teaching device but govern only the device's specific responses to the student. If the same student were to go through the same stupid answers (we'll say) a second time, the teaching machine might well respond in the same way as before. For a human teacher this would be exceedingly

narrow. If a class does not understand a concept the first time, having it explained one way, then the teacher should go at it another way. One of the challenges of the future is that of making the teaching machines learn.

My last comment is really a comment to machine designers. We all now expect computers to talk to each other by telephone. The data phone is an existing thing — not a promise, but a reality. Someone mentioned to me earlier this evening that a computer at the recent Western Joint Computer Conference was operated by remote control through such a data link. Here again you might say we have traversed the full circle, but now it is a helical circle. We are making progress, however. It was at the September 1940 meeting of the American Mathematical Society at Dartmouth College that Dr. George Stibitz of Bell Telephone Laboratories demonstrated a computer that was operated from the Dartmouth academic halls although it was located at the Bell Laboratories. The strange thing to me is that in spite of the fact that everybody has seen telephone communication between computers coming, including the telephone company, you see very little in any present-day computer specifications about the ability of a computer either to place a telephone call or answer a telephone call. To my mind, this is a singular omission. It was easy to predict that computers should have this facility, and it is not a very hard one to provide.

AMDAHL. I am with Dr. Mauchly in being primarily in agreement with Professor McCarthy. I do have some reservations, however, about whether every individual user will desire to operate his own console. Not everyone wishes to operate his own typewriter, for example. I feel that, before it will win common acceptance, the language employed for communication between user and computer will have to become much more universal and efficient, yet be redundant enough to permit varied problem formulation. I believe that typewriter consoles should have larger character fonts to permit ready expression within this language. Some inventions in the areas of keyboards

and print mechanisms to permit character font expansion economically and reliably would be very welcome for computer consoles.

With respect to time-sharing computer systems, Dr. John Cocke of IBM envisages a novel and interesting design for a multiple-usage computer that differs considerably in concept from the one proposed by Professor McCarthy. This is a computer that is one, yet many. It employs a large drum or disc type of storage for the main memory. Also on this disc or drum are arithmetic and address registers for a large number of separate computing units. As the drum or disc turns, these arithmetic and address registers are processed sequentially by a central processor; each one is processed for the execution of a single instruction. Concurrent with the sequential processing, memory references of all computing units are made as the appropriate address angles are passed. Since the time of latency for a memory reference never exceeds the time of latency for the computing unit requesting it, the drum or disc appears to be random-access memory to each computing unit. The central processor employed for the sequential simulation of the computing units does require a high-speed random-access memory adequate to hold all memory reference data for one drum or disc revolution. It could also hold all the individual arithmetic and address registers.

This computer concept is suitable for a large number of low-speed computers all time sharing a central processing unit. It could permit an extremely large and an effectively random-access memory at a very reasonable cost. It does have the disadvantage, however, of not readily permitting the allocation of more than the capacity of a single computing unit to any one problem. This allocation can be performed only for a problem whose formulation permits simultaneous multiple processing of portions of itself.

Professor McCarthy made no allusion to improved order codes for his large private computer. It is my belief that it will soon be possible to design computers whose internal language will be much closer to the programming language employed

than is the case with present-day computers. I think it unlikely that they will reach a one-to-one correspondence, but a higher degree of correspondence would permit much more efficient use of the computer. This is particularly true for programs that are prepared almost entirely to fit the mathematical and physical concepts of the individual user rather than the computing hardware. Compiling programs can be used to transform the user's formulation into a more efficient one for computer processing. For debugging purposes, however, the user would also require a pseudo-inverse transformation to relieve him of considerations of the computer's internal properties.

I stated that computers might soon have order codes more closely related to programming languages. A modest approach to this appears in a logic-processing computer delivered to the Rome Air Development Center in late 1960 or early 1961 by the Aeronutronic Division of the Ford Motor Company. I designed this logic processor in the spring of 1958 in response to the need for evaluating logic equations directly without recourse to a general-purpose computer for prior assembly and compiling. I have since generalized the concepts somewhat so that numeric evaluation of algebraic equations is handled in a completely equivalent fashion. As an algebraic equation is read, the arithmetic operations and the still incomplete set of associated operands are stacked in sequence, and action is deferred until the keystone operand for the locally inner parenthetical expression is sensed. At this time the stacked operands and operations are executed in reverse sequence to the stacking. When the lowest limit of parenthetical enclosure, consistent with the present position in the equation, has been reached, reading of the equation continues as before. Recognition of the equality sign causes the final levels of the deferred operands and operations to be executed, and the final value of the equation is computed.

Such a computer organization is actually quite simple in both concept and hardware. It is only a small step in the direction of designing a problem-oriented order code, but it is significant that one common subset of a mathematical language

has yielded so simply. In my opinion, the invention of further computing techniques that are appropriate to other subsets requires only time and directed effort.

I have some final remarks to make relative to the requirements of the time-sharing and multiprocessing concept favored by Professor McCarthy. In the requirement for relocatability, he suggests that when a program ceases to be active it leaves an odd-sized hole in memory. He also suggests that it must be possible to move other programs down to fill this hole so as to leave a contiguous block of space for new programs and data. It is possible to generalize relocation more completely so that such program and data transfers need not be performed. Rather, the new program and associated data can be fitted into a selection of available odd-sized holes and be processed from this noncontiguous storage. Incidentally, the same feature could provide memory protection automatically. The newly announced Burroughs B–5000 has this capability, if I interpret its description properly. In such a time-sharing computer, progression through a large number of users' programs would soon make memory allocation look like an unplanned patchwork quilt. It would appear completely orderly, however, to each and every active program.

In the multiprocessor-multimemory system suggested, the use of more than one processor depends on the effective memory traffic rate that can be achieved. The possible speed of processors appears to be increasing to the point where several 2-microsecond memories could be exercised quite effectively by a single processor. To achieve this, the complexity of the processor must be greater than that of a processor connected to a single higher-speed memory. It is doubtful, however, that this increase in complexity would be as great as that of several slower processors with a memory-reference arbiter serving them. The single higher-speed central processor also has the advantage of being able to provide the maximum computing capacity to a single active program without requiring this active program to admit to simultaneous processing of parts of itself. The single processor does have a disadvantage, however. If the processor mal-

functions, the entire computing capacity vanishes, not just a given fraction of the capacity. Of course, acquiring the ability to shift computing tasks from a malfunctioning unit to the remaining good units would require a great deal of system-program planning.

McCarthy. What I have to say in reply will deal mainly with Dr. Amdahl's comments. I agree that we need larger character fonts and input-output devices more flexible than typewriters. Even with present typewriters, however, a great deal can be done, and input-output does not seem to be the main limiting factor at present. With regard to John Cocke's computer, I guess I do not fully understand what the relationship is between the capability of such a computer and the capability of the kind of computer that I described. It sounds like less computer for a lot less money.

With respect to the matter of making order codes more like the problem-oriented languages, I disagree with Dr. Amdahl's point of view. It is true that order codes will be improved, but it seems to me that this improvement is not going to reduce by an order of magnitude the time required for a given calculation. I think that reduction by a factor of 2 is as much as can be expected. My reason for believing this has to do with the time consumed by the basic arithmetic operations themselves, and by memory references for instructions and data. When we first started programming the Lisp system for symbolic expressions, we were very enthusiastic about the possible advantages to be gained from new instructions. It turns out that new instructions could do things faster, but not much faster. The way we saw this was by comparing the total memory references that the computer made with the effective or unavoidable memory references. We arrived at an efficiency of about one-sixth for the case of the IBM 709 computer. That is, one-sixth of the memory references were references to actual working data. (Adding references to instructions to references to data would have improved the efficiency figure.) It seemed to us that if we had had the instructions we wanted, we could have doubled

the efficiency. And if we have any say about the order codes of subsequent computers, we certainly shall attempt to achieve this. But I doubt that any larger improvement than this is to be expected, either from order codes resembling problem-oriented languages or from order codes allowing programs which look like algebraic expressions.

Another question on which I disagree somewhat with Dr. Amdahl is whether we have to move programs down to fill odd-sized blocks, or whether we can leave them as they are. The artificial-intelligence group at M.I.T. happens to be the proprietors of a list-type programming system that does not move blocks down but manages to get along very well with odd-sized blocks. This works very well for certain kinds of calculation. However, some rather ridiculous results have been obtained by using the system where it is not suited. An example is matrix calculation. It is true that a matrix can be represented by a list of lists, where the individual lists run all through storage, but addresses must be computed for matrix calculation. If you determine that you want the seventeenth element in a matrix, it unfortunately takes much longer to count seventeen steps down a list (however you do it) than it does to proceed directly to register $A + 17$. From our experience in using unordered systems, we have come to realize that there are certain important classes of calculations in which it is preferable to retain the numerical method of addressing.

My final comment concerns the question of a single high-speed processor versus several processors. It should be understood that the answer to this question depends very much on the current state of technology. For example, if some discovery makes very high speed memory feasible, so that the ratio of memory speed to logical speed becomes larger than it is at present, then a very straightforward computer design would be appropriate. But if the speed of logical operations remains much higher than the speed of memory, a single processor has to look very far ahead in the program to achieve efficiency. The Stretch computer, for example, achieves its speed by looking up to four instructions ahead. Even this is accomplished

with considerable difficulty and some inefficiency. With the present ratio of speed of memory to speed of logic, it might be necessary to accomplish as many as ten or twenty memory references simultaneously. This would require the computer to have still greater powers of foresight.

Amdahl said that the disadvantage of using several processors is that in order to apply all the capacity to a single problem, you have to do some tricky programming; and even then you may not succeed. For a computer that is used by many people, this does not seem to be a serious limitation.

General Discussion

ANONYMOUS. Professor McCarthy, you stated that the memory is relatively more expensive than the central processing unit. But looking at your system organization, the only time sharing you seem to be doing is on the central processing unit and not on the memory. Would you care to comment on that?

MCCARTHY. If you have a very large memory, say on the order of a million memory words, it will consist of a number of boxes containing something like 16,000 words each. I did intend that these boxes be operated simultaneously, so that the memory *would* be time shared, at least in this sense.

ANONYMOUS. But this is not the same kind of time sharing as with your central processing unit. The same processor works on all programs, but the same word of memory is assigned only to one specific program. To infer your memory requirements, you multiply the number of programmers by the average size of a program. To infer the requirements for your central processing unit, you multiply by the average execution time of a program. These are very different things.

MCCARTHY. Are you suggesting that different users share the same memory register? This is possible only by making rapid transfers to and from secondary storage, a procedure that does not seem desirable. By the way, the memory used by system programs *is* time shared. Although there are many processors in the system I described, there is only one set of system programs. One of the flashiest features of the system is that different processors can be executing the same system program at the same time.

ANONYMOUS. Professor McCarthy, are you familiar with the real-time computer of Remington Rand? If so, I want to ask to what degree you think this computer already meets the requirements that you stated. One of its features that you did not mention as being desirable is its ability to accept voice commands. This reduces both the knowledge of special codes which the operator has to have and also the probability of error in input. The Remington Rand computer was designed especially for time sharing and is in use in the Capital Airlines system.

MCCARTHY. I am not familiar with this computer, but I will say one thing nevertheless. It does not have enough memory. As for voice commands, I am in favor of voice input to computers, but as for reducing the amount of information the user has to have, this is not as obvious a virtue as it may seem. To tell the truth, the amount of information one has to acquire in order to program is disgracefully small as it is, compared with the amount of information one has to acquire in order to do well in other fields. What is more important is to extend our ability to use computers, make them do things that we cannot now make them do. As we develop the art of computation, the amount that a user will have to know in order to use computers with full effectiveness should increase rather than decrease.

STEVENS. You have been talking about a central computer and regional consoles. How about the possibility of regional computers and merely a central arbitrator?

MCCARTHY. This question reminds me of the polymorphic computers put forward by Ramo-Wooldridge. I think that such systems are difficult to program, do not make enough memory available to individual users in a straightforward way, and do not allow as efficient sharing of system programs as the multiprocessor system I described.

REYNOLDS. Don't you think that if good input-output pro-

grams were available you could use secondary storage much more effectively and not need such large primary storage? Do you think the large primary storage is really worth the cost that it would incur?

McCarthy. This is a difficult question. Certainly for many purposes a cheaper system could keep the user's program in secondary storage and bring it in as part of each active quantum. Several comments can be made on this possibility. First, very large primary storage is not so terribly expensive and probably will become cheaper. Second, when we get into the new applications of teaching machines, symbolic calculation, and so forth, requests for individual actions are on the average going to require relatively little computation. Thus the cost of bringing in a large program from secondary storage to do a small job is going to loom larger than it does at present. Finally, certain problems now being contemplated do require very large, directly accessible storage.

Fiore. I am going to ask Dr. Mauchly to make any observations that he would care to make at this point.

Mauchly. We have confined our attention so far tonight to the development of the digital computer, but the analog computer also has had considerable development over the same span of years, especially in the engineering fields. One of the things that impresses me about the garden variety of analog computer, if it is at all general purpose and not specifically conceived to do just one job like process control, is that it is very often set up with patch cords, cables, switches, and so forth. Thus problem-setup time is of importance. Even before analog computers went electronic, we had right here at M.I.T. one of the large examples of the trend toward automatic setup: a differential analyzer of the mechanical variety which was set up from punched paper tape. The trend toward automatic setup continues, but there is evidence of an even more alarming (or encouraging) trend. Instead of fighting over the problem of whether a particular calculation should be done by analog methods or digital methods, there is a trend toward combining the two to produce some sort of hybrid unit, with both analog

and digital computers cooperating to solve the same problem. It seems to me that for the kind of situation which Professor McCarthy envisages, a combination like this might be very effective, provided that the analog part could be set up from the remote consoles.

Fiore. Gene [Amdahl], have you any comments?

Amdahl. I should like to pursue Dr. Mauchly's idea a little further. In addition to attaching an analog computer to the system, one might also attach other kinds of digital computers that are individually designed to be particularly good at certain functions. In symbol-manipulating programs, for example, certain features of a digital computer might be heavily used, while other features are not used at all. For instance, the floating-point capabilities may not be used. One readily can imagine a system in which several simple computers, each one slanted toward a particular task, are attached to the same memory. Different portions of the same program then would be executed on different computers. This would be sort of an extension of the Gamma 60 computer system.

Fiore. John [McCarthy], do you want to have a final word before we close the meeting?

McCarthy. Well, it is difficult to find a final word that is somehow worthy of the honor of closing the meeting. All I was going to do was quibble a little more on the last point that was made. The idea of having specialized computers for specialized tasks in order to save money on the hardware seems like a good one, but if we have to program the assignment of different parts of a task to different processors, we had better be sure that the money saved on hardware is not spent on programming.

References

1. Strachey, C., "Time Sharing in Large, Fast Computers," in *Information Processing*, Proceedings of the International Conference on Information Processing, UNESCO, Paris 15–20 June 1959, UNESCO, Paris, 1960, pp. 336–341.

2. Licklider, J. C. R., "Man-Computer Symbiosis," *IRE Transactions on Human Factors in Electronics*, Vol. HFE-1, 4–11 (Mar., 1960).

3. Bush, Vannevar, "As We May Think," *Atlantic Monthly*, Vol. 176, 101–108 (July, 1945).
4. Slagle, J., "A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus, Symbolic Automatic Integrator (SAINT)," Ph.D. thesis, Department of Mathematics, Massachusetts Institute of Technology, May, 1961.

7

A New Concept in Programming

Selected Bibliography

Social Issues and Management Implications

Shultz, G. P., and Whisler, T. L., Eds., *Management Organization and the Computer*, The Free Press of Glencoe, Illinois, 1960.

Simon, H. A., "The Corporation: Will it be Managed by Machines?," *Management and Corporations, 1985*, McGraw-Hill Book Co., New York, 1960.

Simon, H. A., *The New Science of Management Decision*, Harper & Brothers, New York, 1960.

Wiener, N., *The Human Use of Human Beings*, Houghton Mifflin Company, Boston, 1950.

Wiener, N., "Some Moral and Technical Consequences of Automation," *Science*, Vol. 131, 1355–1358 (May 6, 1960).

Simulation of Large Systems

Cohen, K. J., and Cyert, R. M., "Computer Models in Dynamic Economics," *The Quarterly Journal of Economics*, Vol. 75, 112–127 (Feb., 1961).

Conway, R. W., Johnson, B. M., and Maxwell, W. L., "Some Problems of Digital Systems Simulation," *Management Science*, Vol. 6, 92–110 (Oct., 1959).

Forrester, J. W., *Industrial Dynamics*, The M.I.T. Press and John Wiley & Sons, Inc., New York, 1961.

Geisler, M. A., "The Simulation of a Large-Scale Military Activity," *Management Science*, Vol. 5, 359–368 (July, 1959).

Orcutt, G. H., Greenberger, M., Korbel, J., and Rivlin, A. M., *Microanalysis of Socioeconomic Systems*, Harper & Brothers, New York, 1961.

Orcutt, G. H., Shubik, M., Simon, H. A., and Clarkson, G. P. E., "Simulation: A Symposium," three articles in *American Economic Review*, Vol. 50, 894–932 (Dec., 1960).

Pool, I. de S., and Abelson, R., "The Simulmatics Project," *Public Opinion Quarterly*, Vol. 25, 167–183 (Summer, 1961).

Artificial Intelligence

Gelernter, H. L., Hansen, J. R., and Loveland, D. W., "Empirical Explorations of the Geometry Theorem Machine," *Proceedings of the Western Joint Computer Conference*, San Francisco, Calif., May, 1960, The Institute of Radio Engineers, New York, 1960, pp. 143–149.

Minsky, M., "Steps Toward Artificial Intelligence," *Proceedings of the IRE*, Vol. 49, 8–30 (Jan., 1961).

Newell, A., Shaw, J. C., and Simon, H. A., "Chess Playing Programs and the Problem of Complexity," *IBM Journal of Research and Development*, Vol. 2, 330–335 (Oct., 1958).

Samuel, A. L., "Some Studies in Machine Learning, Using the Game of Checkers," *IBM Journal of Research and Development*, Vol. 3, 210–229 (July, 1959).

Selfridge, O. G., and Neisser, U., "Pattern Recognition by Machine," *Scientific American*, Vol. 203, 60–68 (Aug., 1960).

Shannon, C. E., "Programming a Digital Computer for Playing Chess," *Philosophical Magazine*, Vol. 41, 256–275 (Mar., 1950).

Wang, H., "Toward Mechanical Mathematics," *IBM Journal of Research and Development*, Vol. 4, 2–22 (Jan., 1960).

Comparison with Living Organisms

Ashby, W. R., *Design for a Brain*, 2nd ed., John Wiley & Sons, Inc., New York, 1960.

McCulloch, W. S., and Pitts, W., "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, Vol. 5, 115–137 (1943). Articles also in *ibid.*, Vol. 7, 89–93 (1945), and Vol. 9, 127–147 (1947).

Rosenblith, W. A., "The Quantification of the Electrical Activity of the Nervous System," in *Quantity and Quality*, Lerner, D., Ed., The Free Press of Glencoe, Inc. (Crowell-Collier Publishing, Co.), New York, 1961, pp. 87–102.

Turing, A. M., "Can a Machine Think?," in *The World of Mathematics*, Vol. 4, Newman, J. R., Ed., Simon and Schuster, Inc., New York, 1956, pp. 2099–2123.

von Neumann, J., *The Computer and the Brain*, Yale University Press, New Haven, Conn., 1958.

Walter, W. G., *The Living Brain*, W. W. Norton & Company, Inc., New York, 1953.

Wiener, N., *Cybernetics*, 2nd ed., The M.I.T. Press and John Wiley & Sons, Inc., New York, 1961.

Theory of Computation

Davis, M., *Computability & Unsolvability*, McGraw-Hill Book Co., New York, 1958.

McCarthy, J., "A Basis for a Mathematical Theory of Computation, Preliminary Report," *Western Joint Computer Conference*, Los Angeles, Calif., May 9–11, 1961, The Institute of Radio Engineers, New York, 1961, pp. 225–238.

Turing, A. M., "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, Ser. 2, Vol. 42, 230–265 (1937).

von Neumann, J., "The General and Logical Theory of Automata," in *The World of Mathematics*, Vol. 4, Newman, J. R., Ed., Simon and Schuster, Inc., New York, 1956, pp. 2070–2098.

Mechanical Translation

Delavenay, E., *An Introduction to Machine Translation*, Frederick A. Praeger, Inc., New York, 1960.

Edmundson, H. P., Ed., *Proceedings of the National Symposium on Machine Translation*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1961.

Locke, W. N., and Booth, A. D., Eds., *Machine Translation of Languages*, The Technology Press of M.I.T. and John Wiley & Sons, Inc., New York, 1955.

Mechanical Translation (MT), journal published irregularly at M.I.T. (Room 20D–102), Cambridge, Mass.

Oettinger, A. G., *Automatic Language Translation*, Harvard University Press, Cambridge, 1960.

Research on Mechanical Translation, Hearings Before the Special Investigating Subcommittee of the Committee on Science and Astronautics, U.S. House of Representatives Report 2021, 86th Congress, 2nd Session, U.S. Government Printing Office, Washington, D. C., May, 1960.

Information Storage and Retrieval

Brownson, H. L., "Research in Handling Scientific Information," *Science*, Vol. 132, 1922–1931 (Dec. 30, 1960).

Bush, Vannevar, "As We May Think," *Atlantic Monthly*, Vol. 176, 101–108 (July, 1945).

Current Research and Development in Scientific Documentation,

National Science Foundation, U.S. Government Printing Office, Washington, D. C. (published twice yearly).

Documentation, Indexing, and Retrieval of Scientific Information, Staff of U.S. Senate Committee on Government Operations, Senate Doc. 113, 86th Congress, 2nd Session, U.S. Government Printing Office, Washington, D. C., 1960.

Mooers, C. N., "The Next Twenty Years in Information Retrieval: Some Goals and Predictions," *Proceedings of the Western Joint Computer Conference*, San Francisco, 1959, The Institute of Radio Engineers, New York, 1959, pp. 81–86.

Noncouventional Technical Information Systems in Current Use, National Science Foundation, U.S. Government Printing Office, Washington, D. C., No. 1, 1958, No. 2, 1959, and Supplement, 1960.

Proceedings of the International Conference on Scientific Information, 2 vols., National Academy of Sciences and National Research Council, Washington, D. C., 1959.

Swanson, D. R., "Searching Natural Language Text by Computer," *Science*, Vol. 132, 1099–1104 (Oct. 21, 1960).

Taube, M., and Wooster, H., Eds., *Information Storage and Retrieval*, Columbia University Press, New York, 1958.

Time Sharing and Man-Machine Interaction

Cohen, K. J., and Rhenman, E., "The Role of Management Games in Education and Research," *Management Science*, Vol. 7, 131–166 (Jan., 1961).

Licklider, J. C. R., "Man-Computer Symbiosis," *IRE Transactions on Human Factors in Electronics*, Vol. HFE-1, 4–11 (Mar., 1960).

Strachey, C., "Time Sharing in Large, Fast Computers," in *Information Processing*, Proceedings of the International Conference on Information Processing, UNESCO, Paris 15–20 June 1959, UNESCO, Paris, 1960, pp. 336–341.

Teager, H. M., "Systems Considerations in Real-Time Computer Usage," *Proceedings of Conference on Application of Digital Computers to Automatic Instruction*, John Wiley & Sons, Inc., New York, forthcoming 1962.

Programming Languages

Bemer, R. W., "Survey of Modern Programming Techniques," *Computer Bulletin* (British), Vol. 4, 127–135 (Mar., 1961).

COBOL — 1961, U.S. Government Printing Office, Washington, D. C., June 13, 1961.

Gill, S., "Current Theory and Practice of Automatic Programming," *Computer Journal* (British), Vol. 2, 110–114 (Oct., 1959).

McCarthy, J., "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I," *Communications of the Assoc. for Computing Machinery*, Vol. 3, 184–195 (Apr., 1960).

McCracken, D., *A Guide to FORTRAN Programming*, John Wiley & Sons, Inc., New York, 1961.

Naur, P., Ed., Backus, J. W., and others, "Report on the Algorithmic Language Algol 60," *Communications of the Assoc. for Computing Machinery*, Vol. 3, 299–314 (May, 1960).

Orchard-Hays, W., "The Evolution of Programming Systems," *Proceedings of the IRE*, Vol. 49, 283–295 (Jan., 1961).

Ross, D. T., "The Design and Use of the APT Language for Automatic Programming of Numerically Controlled Machine Tools," *Proceedings of the 1959 Computer Applications Symposium*, Armour Research Foundation of Illinois Institute of Technology, Chicago, Ill., 1960, pp. 80–99.

Yngve, V. H., "COMIT as an IR Language," *Communications of the Assoc. for Computing Machinery*, Vol. 5 (Jan., 1962).

Processing and Control

Gotlieb, C. C., and Hume, J. N. P., *High-Speed Data Processing*, McGraw-Hill Book Co., New York, 1958.

Grabbe, E. M., Ramo, S., and Wooldridge, D. E., Eds., *Handbook of Automation, Computation and Control*, John Wiley & Sons, Inc., New York, Vol. 1, 1958, Vol. 2, 1959, Vol. 3, in press.

Gregory, R. H., and Van Horn, R. L., *Automatic Data-Processing Systems*, Wadsworth Publishing Company, Inc., San Francisco, 1960.

Kozmetzky, G., and Kircher, P., *Electronic Computers and Management Control*, McGraw-Hill Book Co., New York, 1956.

Ledley, R. S., *Digital Computer and Control Engineering*, McGraw-Hill Book Co., New York, 1960.

Malcolm, D. G., and Rowe, A. J., Eds., *Management Control Systems*, John Wiley & Sons, Inc., New York, 1960.

Truxal, J. G., "Computers in Automatic Control Systems," *Proceedings of the IRE*, Vol. 49, 305–312 (Jan., 1961).

Conference Proceedings and Special Collections

Alt, F. L., Ed., *Advances in Computers*, Vol. 1, Academic Press, Inc., New York, 1960.

Editors of Scientific American, *Automatic Control*, Simon and Schuster, Inc., New York, 1955.

Information Processing, Proceedings of the International Conference on Information Processing, UNESCO, Paris 15–20 June 1959, UNESCO, Paris, 1960.

Lyapunov, A. A., Goodman, R., and Booth, A. D., *Problems of Cybernetics*, Vol. 1, Pergamon Press, Inc., 1960 (translated from the Russian).

Mechanisation of Thought Processes, 2 vols., Her Majesty's Stationery Office, London, 1959.

Proceedings of the Computer Applications Symposium, Armour Research Foundation of Illinois Institute of Technology, Chicago, 1959, 1960.

Proceedings of the Eastern and Western Joint Computer Conferences, The Institute of Radio Engineers, New York, 1955 and later years.

Shannon, C. E., and McCarthy, J., Eds., Princeton, N.J., *Automata Studies*, Princeton University Press, 1956.

Yovits, M. T., and Cameron, S., Eds., *Self-Organizing Systems*, Pergamon Press, Inc., New York, 1960.

BIBLIOGRAPHIES

Malcolm, D. G., "Bibliography on the Use of Simulation in Management Analysis," *Operations Research*, Vol. 8, 169–177 (Mar.–Apr., 1960).

Minsky, M., "A Selected, Descriptor-Indexed Bibliography to the Literature on Artificial Intelligence," *IRE Transactions on Human Factors in Electronics*, Vol. HFE-2, 39–55 (Mar., 1961).

Shubik, M., "Bibliography on Simulation, Gaming, Artificial Intelligence and Allied Topics," *Journal of the American Statistical Association*, Vol. 55, 736–751 (Dec., 1960).

Index

Martin Greenberger
EDITOR

Computers and the World of the Future

THE M.I.T. PRESS
Massachusetts Institute of Technology
Cambridge, Massachusetts

1962